High-level Pythonic Interface in ecCodes

> Note: The high-level Python interface is currently experimental and may change in a future release. It is provided here to solicit feedback.

New Python Classes

In addition to the low-level C-like interface, there are now Python classes that can be used to work with GRIB and BUFR messages in a more object-oriented manner.

GribFile and **BufrFile**: Classes that implement access to a GRIB/BUFR file. A GribFile/BufrFile closes itself and its messages when it is no longer needed (when used with the context manager)

GribMessage and **BufrMessage**: Classes that implement access to a GRIB/BUFR message. A GribMessage/BufrMessage allows access to the message's key-value pairs in a dictionary-like manner and closes the message when it is no longer needed, coordinating this with its host file.

GribFiles and BufrFiles can be treated mostly as regular files and used as context managers, as can GribMessages/BufrMessages. Each of these classes destructs itself and any child instances appropriately.

GribIndex: A class that implements a GRIB index that allows access to the indexing functionality.

Details

```
# A GRIB file handle meant for use in a context manager.

# One can easily iterate over each message in the file::

# Usage::
>>> with GribFile(filename) as grib:
...     # Print number of messages in file
...     len(grib)
...     # Open all messages in file
...     for msg in grib:
...         print(msg[key_name])
>>> # When the file is closed, any open messages are closed
```

```
# A GRIB message.

# Each ``GribMessage`` is stored as a key/value pair in a dictionary-
like
# structure. It can be used in a context manager or by itself. When the
# ``GribFile`` it belongs to is closed, the ``GribFile`` closes any open
# ``GribMessage``s that belong to it. If a ``GribMessage`` is closed
before
# its ``GribFile`` is closed, it informs the ``GribFile`` of its
closure.

# Scalar and vector values are set appropriately through the same
method.

# ``GribMessage``s can be instantiated from a ``GribFile``, cloned from
# other ``GribMessage``s or taken from samples. Iterating over the
members
# of a ``GribFile`` extracts the ``GribMessage``s it contains until the
# ``GribFile`` is exhausted.

# Usage::
>>> with GribFile(filename) as grib:
...     # Iterate through each message in the file
...     for msg in grib:
...         # Access a key from each message
...         print(msg[key_name])
...         # Report message size in bytes
...         msg.size()
...         # Report keys in message
...         msg.keys()
...         # Set scalar value
...         msg[scalar_key] = 5
...         # Array values are set transparently
...         msg[array_key] = [1, 2, 3]
...         # Messages can be written to file
...         with open(testfile, "w") as test:
...             msg.write(test)
...         # Messages can be cloned from other messages
...         msg2 = GribMessage(clone=msg)
```

```
# A GRIB index meant for use in a context
manager.


Usage::
>>> # Create index from file with
keys
>>> with GribIndex(filename, keys) as
idx:
...     # Write index to
file
...     idx.write
(index_file)
>>> # Read index from
file
>>> with GribIndex(file_index=index_file) as
idx:
...     # Add new file to
index
...     idx.add
(other_filename)
...     # Report number of unique values for given
key
...     idx.size
(key)
...     # Report unique values indexed by
key
...     idx.values
(key)
...     # Request GribMessage matching key,
value
...     msg = idx.select({key: value})
```

```
# A BUFR file handle meant for use in a context manager.

# One can easily iterate over each message in the file::

# Usage::
>>> with BufrFile(filename) as bufr:
...     # Print number of messages in file
...     len(bufr)
...     # Open all messages in file
...     for msg in bufr:
...         print(msg[key_name])
>>> # When the file is closed, any open messages are closed
```

```
# A BUFR message.

# Each ``BufrMessage`` is stored as a key/value pair in a dictionary-
like
# structure. It can be used in a context manager or by itself. When the
# ``BufrFile`` it belongs to is closed, the ``BufrFile`` closes any open
# ``BufrMessage``s that belong to it. If a ``BufrMessage`` is closed
before
# its ``BufrFile`` is closed, it informs the ``BufrFile`` of its
closure.

# Scalar and vector values are set appropriately through the same
method.

# ``BufrMessage``s can be instantiated from a ``BufrFile``, cloned from
# other ``BufrMessage``s or taken from samples. Iterating over the
members
# of a ``BufrFile`` extracts the ``BufrMessage``s it contains until the
# ``BufrFile`` is exhausted.

# Usage::
>>> with BufrFile(filename) as bufr:
...     # Iterate through each message in the file
...     for msg in bufr:
...         # Access a key from each message
...         print(msg[key_name])
...         # Report message size in bytes
...         msg.size()
...         # Report keys in message
...         msg.keys()
...         # Set scalar value
...         msg[scalar_key] = 5
...         # Array values are set transparently
...         msg[array_key] = [1, 2, 3]
...         # Messages can be written to file
...         with open(testfile, "w") as test:
...             msg.write(test)
...         # Messages can be cloned from other messages
...         msg2 = BufrMessage(clone=msg)
```

This feature is a contribution from DWD for which we are very thankful.